

# "Kaiten-Zushi" - Symulacja Restauracji Sushi

---

Autor

---

Michał Walczyk

---

Symulacja wieloprotocowej restauracji sushi z taśmą wykorzystująca mechanizmy komunikacji międzyprocesowej (IPC) w systemie Linux.

---

Użyty kompilator : TORUS

---

## Temat projektu

Temat 1 Restauracja „kaiten zushi”

Program symuluje działanie restauracji z taśmociągami według następujących zasad:

1. **Infrastruktura:** Lokal posiada taśmę o pojemności \$P\$ talerzyków oraz stoliki o różnej wielkości (od 1 do 4-osobowych) oraz łade
  2. **Kliencii Grupy:** Do lokalu przychodzą grupy (1-4 os.). Wymagana jest odpowiednia liczba opiekunów dla dzieci (max 3 dzieci na dorosłego).
  3. **Kolejkowanie:** Klienci pobierają numerki (stolik/lada). Klienci VIP (max 2%) mają bezwzględne pierwszeństwo w zajmowaniu stolików.
  4. **Menu Podstawowe:** Obsługa (Kucharz) losowo umieszcza na taśmie dania w cenach 10, 15, 20 zł.
  5. **Konsumpcja:** Klienci zdejmują talerzyki z taśmy, gdy te znajdują się przy ich stoliku (każdy zjada od 3 do 10 dań).
  6. **Zamówienia Specjalne:** Klienci mogą zamawiać przez tablet droższe dania (40-60 zł), które kucharz przygotowuje dedykowanie dla konkretnego stolika.
  7. **Płatność:** Po zakończeniu posiłku grupa płaci zsumowany rachunek w kasie i opuszcza lokal.
  8. **Sterowanie (Kierownik):** System reaguje na sygnały: przyspieszenie pracy kuchni (x2), zwolnienie (50%) oraz natychmiastową ewakuację.
  9. **Raportowanie:** Po zamknięciu generowane są statystyki: wyprodukowanych dań, sprzedanych posiłków oraz strat (jedzenie, które zostało na taśmie).
- 

## Technologie

- **Język:** C
- **Wątki:** pthreads
- **IPC:** System V (Semafor, Pamięć Współdzielona, Kolejki Komunikatów)
- **Procesy:** Linux (fork, exec, waitpid)

---

## Struktura projektu

```
kaiten-zushi/
├── common.h      # Definicje, struktury, funkcje pomocnicze
├── main.c        # Manager - inicjalizacja i kontrola symulacji
├── kucharz.c     # Proces kucharza - produkcja dań
├── obsluga.c     # Proces obsługi - taśma i płatności
├── kierownik.c  # Proces kierownika - obsługa sygnałów
├── klient.c      # Proces klienta - konsumpcja (wątki)
├── Makefile      # Reguły kompilacji
├── README.md     # Dokumentacja
└── raport.txt    # Generowany raport (po uruchomieniu)
```

---

## Szczegółowy Opis Działania Programu

Poniżej przedstawiono analizę cyklu życia procesów oraz logikę sterowania zasobami.

### 1.Tworzenie i obsługa plików

- [Mechanizm logowania zdarzeń do pliku raportu](#)
- [Otwieranie/Tworzenie pliku raportu](#)

### 2.Zarządzanie Procesami

- [Uruchamianie procesów personelu](#)
- [Uruchamianie procesów klienta](#)
- [Nadzorowanie życia procesów potomnych](#)

### 3.Obsługa sygnałów i sterowanie

- [Rejestracja funkcji obsługi sygnałów u Kucharza](#)
- [Wysłanie sygnału do procesu](#)
- [Wysłanie sygnału do samego siebie](#)

### 4.Pamięć dzielona

- [Alokacja i dołączenie segmentu pamięci współdzielonej](#)
- [Usunięcie segmentu pamięci](#)

### 5.Synchronizacja

- [Utworzenie zestawu semaforów](#)
- [Wykonanie operacji P lub V na semaforze](#)

### 6.Kolejki komunikatów

- [Wysłanie komunikatu do kolejki](#)

- [Odbiór komunikatu z kolejki](#)

## 7. Wątki

- [Utworzenie wątku dla osoby w grupie](#)
- [Oczekiwanie na zakończenie pracy wątku.](#)
- [Synchronizacja dostępu do zmiennych współdzielonych wewnątrz procesu](#)

---

## Mapa synchronizacji

### 1. Zestawienie Semaforów

W projekcie wykorzystano zestaw 11 semaforów (System V) do zarządzania dostępem do zasobów krytycznych oraz kolejkowania klientów.

ID	Typ	Rola w systemie
0	<b>Mutex</b>	Chroni sekcję krytyczną pamięci współdzielonej (modyfikacja taśmy i stolików).
1	<b>Licznikowy</b>	Kolejka oczekiwania VIP dla miejsc przy ladzie (rezerwa logiczna).
2	<b>Licznikowy</b>	Kolejka oczekiwania VIP dla stolików 1-osobowych.
3	<b>Licznikowy</b>	Kolejka oczekiwania VIP dla stolików 2-osobowych.
4	<b>Licznikowy</b>	Kolejka oczekiwania VIP dla stolików 3-osobowych.
5	<b>Licznikowy</b>	Kolejka oczekiwania VIP dla stolików 4-osobowych.
6	<b>Licznikowy</b>	Kolejka oczekiwania Standard dla miejsc przy ladzie.
7	<b>Licznikowy</b>	Kolejka oczekiwania Standard dla stolików 1-osobowych.
8	<b>Licznikowy</b>	Kolejka oczekiwania Standard dla stolików 2-osobowych.
9	<b>Licznikowy</b>	Kolejka oczekiwania Standard dla stolików 3-osobowych.
10	<b>Licznikowy</b>	Kolejka oczekiwania Standard dla stolików 4-osobowych.

**Kod:** [Link do pliku](#)

### 2. Typy Komunikatów (Kolejka komunikatów)

Kolejka komunikatów obsługuje dwa niezależne kanały wymiany informacji, rozróżniane polem mtype:

Typ (mtype)	Stała	Kierunek
1	TYP_ZAMOWIENIE	Klient -> Kucharz
2	KANAL_PLATNOSCI	Klient <-> Obsługa

---

## Testy

Poniżej przedstawiono dowody poprawnego działania systemu w kluczowych scenariuszach.

## T1: Test priorytetów i logiki VIP

- **Cel testu:** Sprawdzamy, czy obsługa klientów VIP działa zgodnie z założeniami, czyli omijają kolejki, nie zajmują miejsca przy ladzie, tylko przy stolikach oraz zajmują cały stół, a nie dosiadają się. Zmniejszamy ilość stołów aby wymusić czekanie i sprawdzenie mechanizmu VIPa
- **Konfiguracja programu:**
  - Zwiększamy szansę na pojawienie się klienta VIP do 50% w `czy_vip`
  - Zmieniamy ilość stołów w define w `common.h` na 2 z każdego typu
- **Oczekiwany wynik:** Oczekujemy zobaczyć w logach programu, że Vip omija kolejki i od razu zajmuje miejsce jeżeli jest wolny stół, Natomiast jeżeli wszystkie stoliki są zajęte VIP czeka w kolejce, ale jest obsługiwany priorytetowo i ma pierwszeństwo nad normalnymi klientami do zajęcia stolika jeżeli jakiś się zwolni
- **Wynik**

```
[VIP 219626] SIADA (Pusty) ID 16 (Cel: 40) | Wiek: 84 5 9 5
[VIP 218860] Zaplacono 255 zł.
[VIP 218082] SIADA (Pusty) ID 17 (Cel: 24) | Wiek: 11 16 18 5
[VIP 219626] Zaplacono 675 zł.
[VIP 218678] SIADA (Pusty) ID 16 (Cel: 12) | Wiek: 21 72 4 6
[KUCHARZ] Specjal 5 dla 218678
[VIP 218678] Zaplacono 195 zł.
[VIP 219240] SIADA (Pusty) ID 16 (Cel: 28) | Wiek: 58 11 10 1
[KUCHARZ] Specjal 4 dla 219240
[VIP 219240] Zaplacono 475 zł.
[VIP 217743] SIADA (Pusty) ID 16 (Cel: 16) | Wiek: 76 6
[KUCHARZ] Specjal 6 dla 217743
[VIP 217743] Zaplacono 235 zł.
[VIP 218956] SIADA (Pusty) ID 16 (Cel: 24) | Wiek: 52 33 56 2
[VIP 218082] Zaplacono 340 zł.
[VIP 218410] SIADA (Pusty) ID 17 (Cel: 18) | Wiek: 17 1
[VIP 218956] Zaplacono 395 zł.
[VIP 219516] SIADA (Pusty) ID 16 (Cel: 40) | Wiek: 65 86 7 7
[KUCHARZ] Specjal 5 dla 219516
[VIP 219516] Zaplacono 690 zł.
[GRUPA 217714] SIADA (Pusty) ID 16 (Cel: 40) | Wiek: 60 58 65 39
[KUCHARZ] Specjal 4 dla 217714
[GRUPA 217714] Zaplacono 610 zł.
[GRUPA 217719] SIADA (Pusty) ID 16 (Cel: 28) | Wiek: 33 77 21 82
[KUCHARZ] Specjal 6 dla 217719
[VIP 218410] Zaplacono 265 zł.
[GRUPA 217722] SIADA (Pusty) ID 17 (Cel: 36) | Wiek: 35 17 75 39
[GRUPA 217719] Zaplacono 470 zł.
[GRUPA 217724] SIADA (Pusty) ID 16 (Cel: 12) | Wiek: 20 2
[KUCHARZ] Specjal 6 dla 217724
[GRUPA 217724] Zaplacono 170 zł.
[GRUPA 217774] SIADA (Pusty) ID 16 (Cel: 14) | Wiek: 17 7
[GRUPA 217774] Zaplacono 210 zł.
[GRUPA 217785] SIADA (Pusty) ID 16 (Cel: 32) | Wiek: 59 4 10 8
[GRUPA 217785] Zaplacono 465 zł.
[GRUPA 217804] SIADA (Pusty) ID 16 (Cel: 24) | Wiek: 44 23 75 8
[KUCHARZ] Specjal 6 dla 217804
[GRUPA 217804] Zaplacono 415 zł.
```

## T2 Test Problem Klient-Kucharz

- **Cel testu:** Sprawdzamy zachowanie systemu w sytuacji, gdy Kucharz jest o wiele szybszy od Klienta. Test ma pokazać, że Kucharz nie nadpisuje dań na taśmie, która jest pamięcią dzieloną tylko czeka, gdy taśma jest pełna.
- **Konfiguracja programu**
  - Ustawiamy rozmiar taśmy `MAX_TASMA` na 1
  - `Odkomentowujemy printf u kucharza`
  - `Odkomentowujemy printf u klienta`
- **Oczekiwany wynik:** Oczekujemy zobaczyć w logach programu, że kucharz zapełnia wszystkie wolne miejsca na taśmie w tym wypadku jedno miejsce, a następnie czeka dopóki klient nie weźmie zamówienia z taśmy. Po zwolnieniu miejsca na taśmie kucharz wraca do pracy do momentu następnego zapełnienia

- **Wynik**

```
[KLIENT 231717] <<< ZDEJMUJE z tasmy (slot 0): Typ 1 (Cena: 10)
[KUCHARZ] >>> KLADZIE na tasme: Typ 2 (Cena: 15)
[KLIENT 231717] <<< ZDEJMUJE z tasmy (slot 0): Typ 2 (Cena: 15)
[KUCHARZ] >>> KLADZIE na tasme: Typ 3 (Cena: 20)
[KLIENT 231717] <<< ZDEJMUJE z tasmy (slot 0): Typ 3 (Cena: 20)
[KUCHARZ] >>> KLADZIE na tasme: Typ 3 (Cena: 20)
[KLIENT 231717] <<< ZDEJMUJE z tasmy (slot 0): Typ 3 (Cena: 20)
[KUCHARZ] >>> KLADZIE na tasme: Typ 1 (Cena: 10)
[KLIENT 231717] <<< ZDEJMUJE z tasmy (slot 0): Typ 1 (Cena: 10)
[KUCHARZ] >>> KLADZIE na tasme: Typ 1 (Cena: 10)
[KLIENT 231717] <<< ZDEJMUJE z tasmy (slot 0): Typ 1 (Cena: 10)
[KUCHARZ] >>> KLADZIE na tasme: Typ 2 (Cena: 15)
[KLIENT 231717] <<< ZDEJMUJE z tasmy (slot 0): Typ 2 (Cena: 15)
[KUCHARZ] >>> KLADZIE na tasme: Typ 3 (Cena: 20)
[KLIENT 231717] <<< ZDEJMUJE z tasmy (slot 0): Typ 3 (Cena: 20)
[KUCHARZ] >>> KLADZIE na tasme: Typ 3 (Cena: 20)
[KLIENT 231717] <<< ZDEJMUJE z tasmy (slot 0): Typ 3 (Cena: 20)
[KUCHARZ] >>> KLADZIE na tasme: Typ 1 (Cena: 10)
[KLIENT 231717] <<< ZDEJMUJE z tasmy (slot 0): Typ 1 (Cena: 10)
[KUCHARZ] >>> KLADZIE na tasme: Typ 1 (Cena: 10)
[KLIENT 231717] <<< ZDEJMUJE z tasmy (slot 0): Typ 1 (Cena: 10)
[KUCHARZ] >>> KLADZIE na tasme: Typ 3 (Cena: 20)
[KLIENT 231717] <<< ZDEJMUJE z tasmy (slot 0): Typ 3 (Cena: 20)
[KUCHARZ] >>> KLADZIE na tasme: Typ 1 (Cena: 10)
[KLIENT 231717] <<< ZDEJMUJE z tasmy (slot 0): Typ 1 (Cena: 10)
[KUCHARZ] >>> KLADZIE na tasme: Typ 1 (Cena: 10)
[KLIENT 231717] <<< ZDEJMUJE z tasmy (slot 0): Typ 1 (Cena: 10)
[KUCHARZ] >>> KLADZIE na tasme: Typ 2 (Cena: 15)
[KLIENT 231717] <<< ZDEJMUJE z tasmy (slot 0): Typ 2 (Cena: 15)
[KUCHARZ] >>> KLADZIE na tasme: Typ 1 (Cena: 10)
[KLIENT 231717] <<< ZDEJMUJE z tasmy (slot 0): Typ 1 (Cena: 10)
[KUCHARZ] >>> KLADZIE na tasme: Typ 3 (Cena: 20)
[KLIENT 231717] <<< ZDEJMUJE z tasmy (slot 0): Typ 3 (Cena: 20)
[KUCHARZ] >>> KLADZIE na tasme: Typ 2 (Cena: 15)
[KLIENT 231751] <<< ZDEJMUJE z tasmy (slot 0): Typ 2 (Cena: 15)
[KUCHARZ] >>> KLADZIE na tasme: Typ 3 (Cena: 20)
[KLIENT 231751] <<< ZDEJMUJE z tasmy (slot 0): Typ 3 (Cena: 20)
[KUCHARZ] >>> KLADZIE na tasme: Typ 2 (Cena: 15)
```

### T3 Test Spójności finansowej

- **Cel testu:** Sprawdzenie czy system nie gubi żadnych zamówień i pieniędzy. Testowana jest poprawność przesyłania komunikatów `msgrcv` i `msgsnd` między procesami klientów a procesem obsługi
- **Konfiguracja programu:**
  - Nic nie zmieniamy, normalne ustawienia
- **Oczekiwany wynik:** W `raport.txt` lub konsoli na końcu w raporcie musi wyjść równość:  
`Wartość produkcji = Utarg + Wartość zmarnowanego jedzenia na taśmie`

- **Wynik:**

```
[MANAGER] Pusto. Raport.

== RAPORT KONCOWY DNIA ==

— RAPORT KUCHARZ (PRODUKCJA) —
Typ 1: 26094 szt. x 10 zł = 260940 zł
Typ 2: 26065 szt. x 15 zł = 390975 zł
Typ 3: 26218 szt. x 20 zł = 524360 zł
Typ 4: 823 szt. x 40 zł = 32920 zł
Typ 5: 851 szt. x 50 zł = 42550 zł
Typ 6: 782 szt. x 60 zł = 46920 zł
LACZNA WARTOSC PRODUKCJI: 1298665 zł

— RAPORT OBSLUGA (SPRZEDAZ) —
Typ 1: 26091 szt. x 10 zł = 260910 zł
Typ 2: 26065 szt. x 15 zł = 390975 zł
Typ 3: 26215 szt. x 20 zł = 524300 zł
Typ 4: 822 szt. x 40 zł = 32880 zł
Typ 5: 850 szt. x 50 zł = 42500 zł
Typ 6: 781 szt. x 60 zł = 46860 zł
LACZNA WARTOSC SPRZEDAZY: 1298425 zł
STAN KASY (UTARG): 1298425 zł

— ZAWARTOSC TASY (STRATY) —
Slot 1: Danie Typ 1 (10 zł)
Slot 2: Danie Typ 6 (60 zł)
Slot 3: Danie Typ 3 (20 zł)
Slot 4: Danie Typ 1 (10 zł)
Slot 5: Danie Typ 1 (10 zł)
Slot 6: Danie Typ 5 (50 zł)
Slot 7: Danie Typ 3 (20 zł)
Slot 8: Danie Typ 3 (20 zł)
Slot 15: Danie Typ 4 (40 zł)
LACZNE STRATY: 240 zł

====
[Kierownik] Koniec zmiany.

[MANAGER] Koniec symulacji.
```

## T4 Test logiki współdzielenia stolików

- **Cel testu:** Sprawdzenie czy semaforey i logika programu dotycząca dosiadania się grup równolicznych do stolików jest poprawna
- **Konfiguracja programu:**
  - Nic nie zmieniamy, normalne ustawienia
- **Oczekiwany wynik** W logu będzie można zobaczyć [GRUPA X] DOSIADA SIE do ID X. Obie grupy jedzą przy stoliku równocześnie, a stolik zajęty jest całkowicie 4/4
- **Wynik**

```
[GRUPA 10635] DOSIADA SIE do ID 25 (Cel: 8) | Wiek: 51 56
[GRUPA 10622] Zaplacono 205 zł.
[GRUPA 10654] DOSIADA SIE do ID 25 (Cel: 8) | Wiek: 27 32
```

---

## Konfiguracja

Użyty kompilator: **TORUS**

### 1.Kompilacja modułów

Projekt można skompilować na dwa sposoby

- wpisując ręcznie komendy:

```
#Klonowanie repozytorium
git clone https://github.com/mwalczyk04/Restauracja-kaiten-zushi.git

#Kompilacja programów
gcc -pthread main.c -o main
gcc kucharz.c -o kucharz
gcc obsluga.c -o obsluga
gcc kierownik.c -o kierownik
gcc klient.c -o klient -pthread
```

- używając gotowego pliku Makefile

```
#Klonowanie repozytorium
git clone https://github.com/mwalczyk04/Restauracja-kaiten-zushi.git

#Kompilacja za pomocą Makefile
make
```

Uruchomienie symulacji: `./main`

### 2.Sygnały

Przed użyciem sygnałów kierownika należy odkomentować [podany fragment](#)

Program na start wypisuje pid kucharza ale można go też sprawdzić przy pomocy: `pgrep kierownik`

Akcja	Syngał
Przyspieszenie	SIGUSR1
Spowolnienie	SIGUSR2
Ewakuacja	Ctrl+C